Fall 2024 - Distributed Database Systems **(CSE 512)**

GROUP PROJECT PROPOSAL

Project Title:

Real-time analytics of Internet traffic flow data

Team Name and Members:

The Web Farm

Kaushik Narayan Ravishankar	1229569564
Lalit Arvind Balaji	1229798494
Anirudh Pramod Inamdar	1229410119
Akash Sivakumar	1229716620

Submission Date: 10-13-2024

1. Introduction

1.1. Background

The Internet has grown into an essential utility for businesses and individuals alike, supporting a wide range of services from communication to entertainment, e-commerce, and more. With the increasing volume of data traversing networks worldwide, monitoring and analyzing traffic patterns has become a critical need for maintaining the health, security, and performance of IT infrastructure. Internet traffic data, which includes information such as source and destination addresses, timestamp, traffic type (e.g., TCP, UDP), and geographical region, holds valuable insights into network behavior. Proper analysis of this data helps organizations in optimizing resources, enhancing security measures, and ensuring better service availability.

To aid in managing these vast amounts of traffic data, visual tools and dashboards have emerged as vital components of network analysis. They allow network administrators to observe traffic trends in real-time, gain insights into regional traffic loads, and make informed decisions on capacity planning, security mitigation, and network efficiency.

1.2. Problem Statement

Despite the critical need for traffic analysis, many organizations lack an easy-to-use solution that can provide both real-time and historical insights into their traffic patterns. Current solutions are often overly complex, expensive, or require extensive technical expertise to implement and maintain. These hurdles make it challenging for teams to effectively monitor and analyze key network data, such as the type of traffic (e.g., TCP/UDP), traffic volume per region, or specific traffic patterns over time (daily, monthly, etc.). Without such capabilities, organizations may struggle with bottlenecks, security vulnerabilities, and inefficient resource allocation.

This project aims to address the gap by providing a simple, yet powerful, user interface that enables users to visualize traffic data and run queries to analyze daily, monthly, or regional traffic patterns. The solution should be both accessible and flexible,

allowing network administrators and analysts to easily gain insights into their internet traffic data.

1.3. Objectives

This project aims to develop a real-time traffic flow analytics system on top of a distributed database architecture. The system should be able to efficiently ingest, store, and query large volumes of network traffic data, and demonstrate concepts such as partitioning, replication, and fault tolerance. Some specific objectives include:

- *Real-time data ingestion*: Setting up a pipeline to process and store continuous streams of network traffic data.
- *Efficient data storage and querying*: Implementing a schema optimized for traffic flow analysis, with compression techniques tailored to different data types.
- *Scalability and performance*: Demonstrating the system's ability to scale horizontally while maintaining performance in querying and data processing.
- *Traffic flow analysis*: Enabling detailed analysis of traffic patterns and behaviors, providing insights that can be visualized and inferred from the data.

2. Project Description

2.1. System Design

This system will ingest data from real-time data streams, process them and store the data in a distributed database system, and provide a user interface for data analysis and visualization. Working under the assumption that the use-case isn't mission-critical, some degree of reliability can be deprioritized. In terms of optimization and configuration, the focus would be on ensuring scalability and high performance for large data volumes.

- *Data Partitioning*: Several factors are taken into account, such as:
    - Region - Grouping ingested data by originating region helps localize queries and optimize data retrieval by reducing network latency.

    - Time - Traffic patterns are time-sensitive. Partitioning by time (e.g., by hour, day) optimizes both real-time queries and historical analyses by limiting the query scope to relevant partitions.
    - Traffic type - Segmenting by traffic type (protocol, volume, etc.) optimizes queries that are interested in protocol-specific trends.
- *Replication Strategies and Fault Tolerance*:
    - Since the data volume is very large, partial replication would suffice to ensure high availability while maintaining the desired performance. A fairly low replication factor such as 2 or 3 would be suitable.
    - Multi-master (N-way) allocation model is a desirable strategy in this situation to ensure real-time updating and high availability, as opposed to other models like master-slave.
- *Query Processing* - Cost factors to consider for optimization include:
    - Data Distribution - this includes network I/O and latency costs, and is the major bottleneck. Locality and pushdown filtering (applying filters close to the data source) help in this aspect.
    - Other potential optimizations include accounting for slower storage mediums, and optimizing CPU utilization.

We plan to provide a simple web-based user interface for data visualization and inference.

2.2. Implementation Plan

The project will be implemented using:

- Programming Languages and Frameworks:
    - Python - kafka-python[1] library for managing Kafka producers and topics, as well as streaming the data to Kafka
    - SQL - manually querying ClickHouse
    - NodeJS - ReactJS and react-apexcharts[2] for visualizing the data, clickhouse-js[3] client library for programmatically interfacing with ClickHouse

- Databases: ClickHouse[4] will be used as the core distributed database system
- Other tools:
    - Apache Kafka[5] - event streaming platform, for real-time data ingestion.
    - Docker - environment setup and easy replication across nodes.
    - ClickHouse Keeper (part of ClickHouse) - coordination of distributed systems. Similar to Apache ZooKeeper[6]. Used for coordination of replication and fault tolerance.

2.3. Data Strategy

The data used for this project will consist of real-time traffic flow records; the format will resemble NetFlow records, containing only metadata from the network and transport layers, such as source and destination IP addresses, port numbers, protocol types, and timestamps. The proposed schema is simple and primarily consists of the following tables:

1. Traffic Table: The fact table that stores flow-level data:
    - `timestamp`: The time at which the flow was recorded.
    - `source_ip` and `destination_ip`: IP addresses involved in the traffic.
    - `source_port` and `destination_port`: The ports used for communication.
    - `protocol_id`: The protocol used for the flow (e.g., TCP, UDP).
    - `bytes_transferred`: The total amount of data transferred during the flow.
2. Geolocation Table: The dimension table that provides enriched information by mapping IP addresses to geographic locations. It will include fields such as:
    - `ip_range_start` and `ip_range_end`: Defining IP ranges.
    - `country`, `region`, and `city`: Geographical details based on IP lookup.

Note: This schema is subject to change during implementation. It is also easily extensible and customizable based on the data available and features needed. For data analysis, filtering can be done based on specific criteria such as timeframe and location. This ensures the dataset is manageable and focused.

Data sourcing and privacy:

- We plan to source data from the vast dataset of packet traces publicly provided by the MAWI Working Group of the WIDE Project[7]. Alternative sources of similar quality are also available, such as CAIDA[8].
- These datasets have been scrambled at the source to eliminate confidential information - in this case, obfuscation of the IP addresses.
- It is also possible to randomly generate synthetic data to augment the data if needed for testing purposes.

3. Methodology

3.1. Technique 1 - Sharding for Partitioning

When data is ingested into a distributed table using Kafka, ClickHouse handles the data distribution seamlessly across multiple shards, located on different nodes within the cluster. This distribution process relies on the sharding key configuration, defined when creating the distributed table, to determine how data partitions are allocated to specific shards.

3.2. Technique 2 - Replication

Each shard is configured using the ReplicationMergeTree engine[9] with a replication factor of 2, meaning that two identical copies, or replicas, of each shard's data are stored on separate hosts within the cluster. This redundancy ensures data integrity and high availability, as each replica can serve as a backup in case of hardware failure or network issues.

When changes occur in a shard, such as data updates or new entries, the replicas are automatically synchronized to reflect these changes.

This synchronization is managed by ClickHouse Keeper, the service responsible for coordinating and maintaining consistency across replicas in the cluster. These Keeper nodes monitor all replicas for consistency, ensuring that each remains up-to-date with the primary shard. In the event of node or shard failure, the Keeper facilitates automatic failover to a replica, allowing uninterrupted data access and maintaining service continuity.

3.3. Technique 3 - Scalability by autonomous Sharding

We can dynamically scale the ClickHouse database by automatically adding new shards based on parameters like region. This approach allows data specific to new parameter values (e.g., new regions) to be stored on dedicated hosts, increasing horizontal scalability. This setup enhances data storage, improves query performance through parallel processing, and strengthens load balancing and fault tolerance with replicas.

Steps to Add New Shards:

1. *Provision New Nodes*: Set up new servers with ClickHouse.
2. *Update Cluster Configurations*: Modify `config.xml` on all nodes to include the new shard and its replicas.
3. *Expand Distributed Table Schema*: Adjust distributed table definitions to direct new parameter data to the added shards.
4. *Restart and Verify*: Restart nodes sequentially to apply changes and verify that new shards function correctly.

Since each shard holds data for a specific parameter value, the need to redistribute existing data is removed, simplifying the scaling process.

3.4. Technique 4 - Materialized Views

Materialized views[10] in ClickHouse are database objects that store the results of a query physically on disk, allowing for faster access to precomputed data. They automatically update as new data arrives, making them particularly useful here, as rapid query performance on large datasets is critical. Materialized views can pre-aggregate data such as hits, unique visitors, or session durations, significantly improving query performance by reducing the need to scan large volumes of raw data.

3.5. Technique 5 - TTL

ClickHouse's TTL (Time to Live) feature[11] enables automated data lifecycle management by specifying how long data should reside in different storage tiers. The hot/warm/cold architecture leverages TTL and storage policies to efficiently manage data based on its age and access frequency.

Overview of Hot/Warm/Cold Architecture in ClickHouse:

- Hot Data: Recent data, frequently accessed. Stored on high-performance storage like SSDs for fast read/write operations.
- Warm Data: Data that is older and accessed less frequently. Moved to moderately fast storage.
- Cold Data: Historical data, rarely accessed. Stored on cost-effective, slower storage like HDDs.

Implementation Steps:

- Define Storage Policies: Create a storage policy that includes multiple volumes representing different storage tiers.

- Set Up Tables with TTL Expressions: Apply TTL rules to move data between storage tiers based on its age.

We can utilize ClickHouse's TTL feature and storage policies and implement a hot/warm/cold architecture that optimizes data storage and access in real-time web traffic analysis. This approach ensures high performance for recent data, cost-effective storage for older data, and automated lifecycle management, enhancing the overall efficiency of the analytics system.

3.6. Technique 6 - Data Compression Strategies

ClickHouse's compression features enable significant storage savings and performance improvements. By applying various column-level compressions and encodings like ZSTD[12], Delta (consecutive difference-based) and Gorilla[13], we can efficiently store various data types such as timestamps, addresses, etc. For example, using the Gorilla or Delta codec on sequential timestamps reduces storage space by storing differences between values, while ZSTD offers higher compression ratios for strings with repeating patterns - in this case, IP addresses.

Implementing these compression techniques reduces the data footprint on each node, optimizing disk usage and network bandwidth—critical for scalability in distributed databases. Compressed data leads to faster read operations due to reduced I/O. Additionally, smaller data sizes facilitate quicker replication and recovery processes, improving fault tolerance.

4. Evaluation Plan

4.1. Metrics for Evaluation

The following metrics will assess the system in terms of common characteristics of distributed database systems, as well as with regard to system performance.

- *Query execution time:* It directly reflects the system's ability to provide timely insights essential for monitoring user behavior and making rapid decisions. It helps identify performance bottlenecks, optimizes resource utilization, and assesses scalability by revealing how well the system handles increased data and user load across distributed nodes.
- *Throughput:* It measures the system's ability to handle high volumes of data ingestion and query processing efficiently. It reflects how well the system scales under load, ensuring timely insights by indicating the amount of data processed or the number of operations completed per unit of time. Throughput can be measured by tracking data ingestion rates and query execution rates using built-in database tools, external monitoring solutions and conducting load tests.

$$Throughput \ = \ Number\ of\ transactions\ /\ time\ (seconds)$$

- *Scalability:* It measures the system's ability to efficiently handle increasing data volumes and user requests without compromising performance or reliability. To measure scalability, we test horizontal scalability (adding more nodes), analyzing how the system performs under increased load, identifying bottlenecks, and optimizing configurations.
- *Availability/fault tolerance:* It measures the system's ability to continue operating during failures such as node crashes, network issues, or hardware malfunctions. To measure fault tolerance, we simulate various failure scenarios (e.g., node failures, network partitions, disk issues) and assess the system's response by monitoring recovery time, data consistency, throughput, and latency under stress. Key metrics include Mean Time to Recovery (MTTR),

availability percentage, and error rates during failures, all of which help identify weaknesses and optimize the system's resilience.

- *Resource utilization:* It assesses how efficiently the system uses CPU, memory, disk, and network resources to handle data and queries. Utilizing ClickHouse's system tables and external tools like Prometheus or Grafana allows for tracking utilization trends, setting thresholds, and configuring alerts for potential issues.

4.2. Expected Outcomes

We expect the outcome of our project work to be a system for real-time web traffic analysis that has low query processing time, encompassing characteristics such as high data freshness and as low resource utilization as possible. We expect our distributed database system to have low latency and high throughput in order to enable high-speed seamless user experience. This would also result in a highly scalable as well as fault tolerant database. We aim to ensure our database is highly redundant and immune to node failures of high-degree impact. Finally, we also expect to produce a user interface that is neat, understandable, with enough information that is visually appealing.

5. Timeline

Milestone	Start Date	End Date	Team Member Responsible For
System Design and Architecture	10/14/2024	10/20/2024	Kaushik
Implementing Distributed	10/21/2024	11/03/2024	Kaushik, Akash

Database and Data Ingestion			
Frontend Development (User Interface)	10/14/2024	11/03/2024	Kaushik, Lalit, Anirudh
Backend Development (Connect UI to Database)	11/04/2024	11/17/2024	Kaushik, Akash
Database Performance Testing	11/18/2024	11/24/2024	Kaushik, Lalit
System Performance Testing	11/18/2024	11/24/2024	Kaushik, Anirudh
Error Fixes, Integration Fixes, Final Round of Testing (if necessary), Report Writing	11/25/2024	12/01/2024	Kaushik, Lalit, Anirudh, Akash

6. Conclusion

This project aims to develop a real-time traffic flow analytics system using a distributed database, with a focus on scalability, performance, and efficient data processing. By leveraging a simple yet extensible schema, incorporating real-time data ingestion, and effectively using compression strategies, the system will showcase key

aspects of distributed database management, such as partitioning, replication, and query optimization. The flexibility of the system allows for detailed traffic flow analysis and exploration, highlighting the benefits of a distributed architecture in handling large data volumes in real-time. This project will serve as a proof-of-concept, demonstrating the practical applications of distributed systems in modern network analytics.

References

[1] dpkp, “GitHub - dpkp/kafka-python: Python client for Apache Kafka,” *GitHub*. Available: https://github.com/dpkp/kafka-python. [Accessed: Oct. 13, 2024]

[2] apexcharts, “GitHub - apexcharts/react-apexcharts: React Component for ApexCharts,” *GitHub*. Available: https://github.com/apexcharts/react-apexcharts. [Accessed: Oct. 13, 2024]

[3] ClickHouse, “GitHub - ClickHouse/clickhouse-js: Official JS client for ClickHouse DB,” *GitHub*. Available: https://github.com/ClickHouse/clickhouse-js. [Accessed: Oct. 13, 2024]

[4] ClickHouse, “GitHub - ClickHouse/ClickHouse: ClickHouse® is a real-time analytics DBMS,” *GitHub*. Available: https://github.com/ClickHouse/ClickHouse. [Accessed: Oct. 13, 2024]

[5] apache, “GitHub - apache/kafka: Mirror of Apache Kafka,” *GitHub*. Available: https://github.com/apache/kafka. [Accessed: Oct. 13, 2024]

[6] apache, “GitHub - apache/zookeeper: Apache ZooKeeper,” *GitHub*. Available: https://github.com/apache/zookeeper. [Accessed: Oct. 13, 2024]

[7] K. Cho, K. Mitsuya, and A. Kato, “Traffic Data Repository at the WIDE Project,” in *2000 USENIX Annual Technical Conference (USENIX ATC 00)*, San Diego, CA: USENIX Association, 0 2000. Available: https://www.usenix.org/conference/2000-usenix-annual-technical-conference/traffic-data-repository-wide-project

[8] CAIDA, “Anonymized Two-Way Traffic Packet Header Traces (2024),” *CAIDA*, 2024. Available: https://catalog.caida.org/dataset/passive_2024_pcap_100g. [Accessed: Oct. 13, 2024]

[9] “Data Replication,” *ClickHouse Docs*. Available: https://clickhouse.com/docs/en/engines/table-engines/mergetree-family/replication. [Accessed: Oct. 13, 2024]

[10] “Materialized View,” *ClickHouse Docs*. Available:

https://clickhouse.com/docs/en/materialized-view. [Accessed: Oct. 13, 2024]

[11] “Manage Data with TTL (Time-to-live),” *ClickHouse Docs*. Available: https://clickhouse.com/docs/en/guides/developer/ttl. [Accessed: Oct. 13, 2024]

[12] facebook, “GitHub - facebook/zstd: Zstandard - Fast real-time compression algorithm,” *GitHub*. Available: http://github.com/facebook/zstd. [Accessed: Oct. 13, 2024]

[13] “Column Compression Codecs - Gorilla,” *ClickHouse Docs*. Available: https://clickhouse.com/docs/en/sql-reference/statements/create/table#gorilla. [Accessed: Oct. 13, 2024]